Efficient Mining of Temporal Safety Properties for Intrusion Detection in Industrial Control Systems

Oualid Koucham, Stéphane Mocanu, Guillaume Hiet, Jean-Marc Thiriet, Frédéric Majorczyk

Efficient Mining of Temporal Safety Properties for Intrusion Detection in Industrial Control Systems

Oualid Koucham * **Stéphane Mocanu** ** **Guillaume Hiet** ***
Jean-Marc Thiriet * **Frédéric Majorczyk** ****

* *U.G.A, CNRS, Gipsa-lab, ({oualid.koucham, jean-marc.thiriet}@grenoble-inp.fr).*
** *U.G.A, CNRS, G-INP, Inria, LIG, stephane.mocanu@imag.fr*
*** *CentraleSupélec, Inria, CNRS (guillaume.hiet@centralesupelec.fr)*
**** *DGA, Inria (frederic.majorczyk@supelec.fr)*

Abstract: Sophisticated process-aware attacks targeting industrial control systems require adequate detection measures taking into account the physical process. This paper proposes an approach relying on automatically mined process specifications to detect attacks on sequential control systems. The specifications are synthesized as monitors that read the execution traces and report violations to the operator. In contrast to other approaches, a central aspect of our method consists in reducing the number of mined specifications suffering from redundancies. We evaluate our approach on a hardware-in-the-loop testbed with a complex physical process model and discuss our approach's mining efficiency and attack detection capabilities.

Keywords: Control and security for critical infrastructure systems, safety and security of CPS, integrated monitoring, intrusion detection, specification mining

1. INTRODUCTION

Safety within industrial control systems (ICS) has classically focused on accidental risks with potential impacts on the system and its environment. Nowadays, we must also take into account the increasing risks originating from malicious intent. In particular, ICS need to cope with the prospect of targeted attacks that rely on an advanced knowledge of the physical process. The lack of sufficient protection mechanisms motivates the need to develop suitable detection mechanisms to identify the occurrence of process-oriented malicious behavior.

ICS rely on both continuous and sequential control to achieve their industrial objective. An attacker can target any aspect to drive the physical process towards a forbidden state. We present an efficient approach to detect process-aware attacks targeting the sequential aspect of ICS. We focus on *sequence attacks* which threaten the process through a malicious temporal ordering of individually legitimate commands. Detection approaches aimed at sequence attacks are classified either as cyber oriented relying on network exchanges between ICS nodes, or as process oriented relying on the state and evolution of physical process variables. Since the finality of sequence attacks is putting the physical process in a forbidden state, a process oriented detection appears to be more suitable. Moreover, such approaches produce violations that can be directly interpreted in terms of actual physical process states rather than low level network messages. This in turn eases the identification and treatment of false alerts for operators familiar with the physical process.

This paper extends and builds on the process oriented approach presented in Koucham et al. (2016). Malicious behavior is detected through violations of process specifications, which consist in temporal safety properties over the states of sensors and actuators. The process specifications are mined from execution traces of the physical process and then synthesized as monitors that report violations.

Our focus in this paper is to limit the overwhelming number of process specifications usually produced by generic mining approaches. We observe that a significant number of these mined specifications exhibit redundancies and show that their elimination leads to a substantial improvement in the number of mined properties. In practice, limiting these redundancies is crucial in order to provide operators with pertinent and manageable alerts which ultimately translates into quicker reactions.

In summary, we first propose an efficient approach to infer and monitor process specifications on the highly structured traces of sequential control systems. We then evaluate and analyze our approach using an experimental testbed in a hardware-in-the-loop setting under both attacks and legitimate manipulations.

The paper is organized as follows. Section 2 provides the necessary background information, reviews related work on intrusion detection, specification mining, and discusses links with fault diagnosis. Section 3 presents our detection approach. Section 4 describes our evaluation setup and implementation. Section 5 provides an analysis of the results. Finally, Section 6 concludes the paper.

2. BACKGROUND AND RELATED WORK

A *threat model* makes explicit the position of the attackers regarding the system (internal or external), their capacities, the targeted assets and the nature of the attacks. In our model, the attacker is positioned within the supervisory zone (internal) and can carry attacks through supervisory ICS workstations that can access the PLCs. We assume that the attacker has the knowledge and the capacities to forge or modify commands aimed at the PLCs to change the state of actuators.

In this paper, we deal with sequence attacks. We focus on this class of sophisticated process-oriented attacks since it has been identified as an important threat to control systems in operational plants (see Robert T. Marsh (1997); Larsen (2008)). Such attacks aim at putting the physical process in a forbidden state through a malicious ordering of individually legitimate commands. In particular, we focus on sequence attacks that modify the relative order of the commands (for instance, sending two consecutive commands to simultaneously open two valves). We do not cover attacks which manipulate the timing between consecutive commands. Detecting such attacks requires more elaborate quantitative models which are beyond the scope of this paper. We also do not cover physical attacks or false data injection attacks involving corrupted sensor data. Since we monitor the system using supervisory flows, detecting such attacks is difficult.

The task of an *intrusion detection system* (IDS) is to automatically detect violations of the security policy of a monitored system. In this paper, we implement an anomaly-based IDS which reports violations as deviations from a reference behavior. Compared to other approaches, such an IDS has the ability to detect novel attacks. On the other hand, the amount of false alerts can be significant if the reference behavior is not complete. By eliminating redundant process specifications, we limit the number of raised false alerts. Among intrusion detection approaches for ICS, the works closest to ours target process-aware attacks as in Caselli et al. (2015); Mitchell and Chen (2014); Carcano et al. (2011). In contrast to Caselli et al. (2015), we focus on process variables instead of network communications since attacks targeted towards the physical process can be best detected and understood by monitoring the evolution of process variables. We use a more expressive formalism for the monitored properties than Mitchell and Chen (2014) and Carcano et al. (2011) and avoid the need to manually specify the properties.

Our detection approach builds on *runtime verification*, which is a lightweight verification technique that checks whether the current execution of the system satisfies or violates a certain correctness property (Leucker and Schallhart (2009)). In our case, the correctness property specifies the temporal ordering of sensor and actuator states. We call monitors the devices which read the execution trace of the system, and return a verdict indicating the status of the property. As discussed in Koucham et al. (2016), linear temporal logic (LTL), introduced in Pnueli (1977), is a suitable formalism for representing the ordering constraints violated by sequence attacks. For instance, the LTL formula $\Box\neg(valve_1 \wedge valve_2)$ is a property stating that $valve_1$ and $valve_2$ should never be simultaneously open. Here, $\neg, \wedge$ are the propositional negation and conjunction operators and $\Box$ is the temporal logic operator corresponding to a global constraint over all subsequent positions in the trace, i.e $\Box\phi$ is true if the formula ϕ is true in the current and all subsequent positions. In our approach, monitors are synthesized as finite state automata from LTL formulae (see D'Amorim and Roşu (2005)).

Applications of runtime verification to the failure diagnosis was explored by Bauer et al. (2006) and Dubey et al. (2011). In Bauer et al. (2006), failure detection is performed by a set of monitors, while failure identification relies on aggregating the verdicts emitted by the distributed monitors and inferring possible causes of failure through a consistency-based diagnosis approach. The system's description is reduced to checking the input-output correctness behavior of the components using monitors. This yields a tractable diagnosis approach in terms of satisfiability in propositional logic. Our solution can be used in conjunction with such a diagnosis approach by automatically inferring and limiting the number of properties to monitor at the input-output ports of a set of PLCs.

Concerning *specification mining*, Lemieux et al. (2015) discuss several techniques to reduce the complexity of the task. However, the number of valid mined specifications can still remain significant. In this paper, we leverage the regular structure of execution traces in sequential control systems to limit the number of infered properties.

3. APPROACH

3.1 General overview

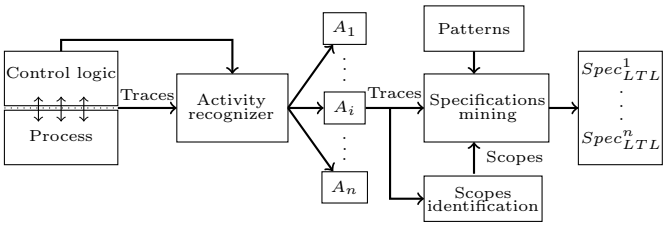

Fig. 1. General overview of the mining approach

In this section, we present a general overview of our approach which spans two phases : (i) a mining phase where process specifications are infered from execution traces of the physical process, and (ii) a monitoring phase where the infered properties are deployed to detect violations.

Mining phase Figure 1 shows the general overview of the mining phase. At first, we collect attack-free execution traces that record the evolution of sensors and actuators states throughout the execution of control logics. The traces are collected from supervisory flows between the supervisors and the PLCs. We focus in our approach on control logics implemented as Sequential Function Charts (SFC) (see John and Tiegelkamp (2010)) as they are particularly suitable for sequential processes.

We distinguish between the control flow of an SFC, represented by selection and parallel branches, and linear step-transition sequences without branching and parallel execution which we call *activities*. We use the SFC's structure to

divide the execution traces per activity (performed by the activity recognizer block in Figure 1), and mine process specifications per activity. The activity recognizer reads the execution traces and uses the SFCs to identify the current active steps and thus the current activities. We focus on activities as they represent the actual sequencing of actions performed by the PLCs, while control flow decides which activities are to be executed.

Our goal is to automatically find safety temporal specifications that are valid on attack-free execution traces of the system. This problem is known as specification mining. Specification mining approaches typically use specification patterns to find the properties that are valid on the traces. For instance, consider the pattern *A never occurs between B and C*. Here, A, B and C are variables which take values in a set of events that might correspond to changes in the states of actuators and sensors. In particular, B and C define the *scope* of the property, i.e the subsequences of the execution trace where the constraint applies, and A specifies the event which never occurs within the scope. If $B \in \{e_0, e_1, e_2, e_3\}$, $C \in \{e_0, e_1, e_2, e_3\}$, and $A \in \{e_4\}$, then possible instantiations of the pattern include : (i) e_4 *never occurs between* e_0 *and* e_2, (ii) e_4 *never occurs between* e_1 *and* e_3, (iii) e_4 *never occurs between* e_1 *and* e_2. The objective is to find all instantiations that are valid on the attack-free traces.

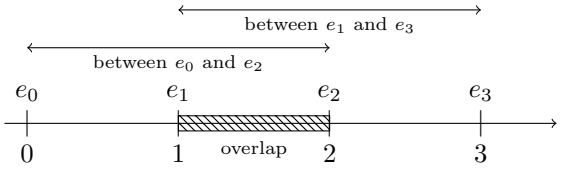

Fig. 2. Scopes overlap illustration

However, a common issue with specification mining approaches is the significant number of resulting valid instantiations. We argue that many of these instantiations are redundant due to scope overlaps. An overlap occurs between two scopes when they delimit intersecting subsequences of the execution trace. We illustrate this situation on the previous example using the execution trace in Figure 2. Each position is associated with an event and the figure shows the subsequences corresponding to the scopes *between* e_0 *and* e_2 and *between* e_1 *and* e_3. These scopes overlap since the subsequences they delimit intersect on the subsequence spanning positions 1 and 2.

To illustrate how such redundancies impact the number of violations sent to an operator, suppose that the properties (i), (ii) and (iii) above are valid. If an attack causes event e_4 to happen between position 1 and 2, property (iii) would be violated. However, due to redundancy, properties (i) and (ii) would also be violated. Thus, instead of a single violation, the operator would need to deal with three redundant violations. This problem worsens as the number of events and scopes become large as in the case of long activities. The operator then risks becoming overwhelmed with the significant number of violations. In this paper, we avoid redundancies through a pre-selection of scopes before mining. Only instantiations on the pre-selected scopes are tested for validity. This pre-selection is done by the scopes identification block in Figure 1.

To determine the valid instantiations, we synthesize each instantiation as a finite state automaton and run the attack-free execution traces. If a violation is raised, we discard the instantiated property. Otherwise, we keep the property to be used during the detection phase. As an output, we obtain a number of LTL process specifications which will be deployed as monitors.

Monitoring phase During the monitoring phase, the deployed monitors return violations which are presented to the process operator. The activity recognizer is also used in this phase to direct the execution traces to the proper monitors. Upon reception of a violation, the operator decides whether it corresponds to a false or a true alert given the activity, the scope and the impacted actuator.

3.2 Fundamental definitions

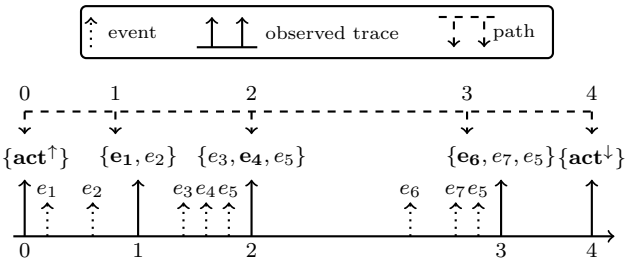

Fig. 3. Relation between events, observed traces and paths

We define AP to be a finite set of boolean variables where each variable refers to a state of a component (sensor or actuator) in the system. Components with continuous values are discretized and mapped to boolean variables. For instance, a level sensor tl reporting continuous values can be discretized to boolean states ($tl \geq th$) and ($tl < th$) with th a threshold. *Events* are changes in the states' values. For a state $a \in AP$, $a^{\uparrow}$ (resp. $a^{\downarrow}$) corresponds to a rising edge (resp. falling edge) of state a.

From the set AP, we generate the set of all possible component events $E = \bigcup_{a \in AP} \{a^{\uparrow}, a^{\downarrow}\}$. We refer to the power set of E as $\mathcal{P}(E)$.

Moreover, we also define boolean variables $act \in Act$, which indicate whether the activity act is currently active. To mark the beginning (activation of the first step) and the end (deactivation of the last step) of activity act, we use a pair of events $(act^{\uparrow}, act^{\downarrow})$. We thus define the set $E_{Act} = \bigcup_{act \in Act} \{act^{\uparrow}, act^{\downarrow}\}$.

Definition 1. (Observed trace). An *observed trace* t of size $n \in \mathbb{N}^*$ over an activity act is an application :

$$t : \{0, 1, \ldots, n-1\} \rightarrow \{act^{\uparrow}, act^{\downarrow}\} \cup \mathcal{P}(E)$$

$$\begin{cases} t(0) = \{act^{\uparrow}\}, t(n-1) = \{act^{\downarrow}\} \\ \forall 0 < i < n-1, t(i) \ \cap \{act^{\uparrow}, act^{\downarrow}\} = \emptyset \end{cases}$$

Our view of the process is based on supervisory trafic with periodic sampling of actuator and sensor states. Thus, events are observed at some discrete points in time. An *observed trace* maps each observation point to a set of

observed events. An observed trace necessarily begins with the activation of an activity and ends with its deactivation. These activity-specific events do not occur anywhere on the trace. Figure 3 provides an example of an observed trace of size 4 over an activity *act*. Each position in the observed trace corresponds to a set of observed events. For instance, events $\{e_1, e_2\}$ are mapped to position 1 while events $\{e_3, e_4, e_5\}$ are mapped to position 2.

In what follows, we consider t to be an observed trace of size $n \in \mathbb{N}^*$ over an activity *act*. We refer to the domain $\{0, 1, \ldots, n-1\}$ of t as $Dom(t)$.

Recall that our objective is to reduce the number of mined process specifications by avoiding scope overlaps. To characterize scope overlaps, we formally introduce the notion of scope and scope cover.

Definition 2. (Scope). A *scope* is a pair of events (e_1, e_2) where $e_1, e_2 \in Act \cup E$

Definition 3. (Scope cover). Let $s = (e_1, e_2)$ be a scope. The cover of s over t, denoted by $C_t(s)$, is the set :

$$C_t(s) = \{(i, j) \in Dom(t)^2 | i < j \wedge e_1 \in t(i) \wedge e_2 \in t(j)\}$$

The cover of a scope returns all the pairs of ordered time positions in the observed trace where the events making up the scope occur. For instance, in Figure 3, the scope (e_1, e_6) has cover $C_t((e_1, e_6)) = \{(1, 3)\}$ since e_1 occurs at position 1 and e_4 occurs at position 3.

Remark 4. (Size of scope cover). Given a scope s, the size of its cover $|C_t(s)|$ can be greater than 1. For instance, in Figure 3, $C_t((e_1, e_5)) = \{(1, 2), (1, 3)\}$

To mine valid properties, we focus in this paper on the absence and universality scope-based LTL specification patterns introduced in Dwyer et al. (1999). These patterns refer to recurring specifications which have been identified through an extensive review of the literature, and hierarchically organized to ease their reuse and interpretation.

Definition 5. (Absence and universality properties). Let S be a set of scopes, $AP_{Ac} \subset AP$ the set of actuator states, and $E_{Ac} \subset E$ the set of actuator events.

- An *absence property* is a predicate $abs(e_1, e_2, ea)$ where $(e_1, e_2) \in S, ea \in E_{Act} \cup AP_{Ac}$
- A *universality property* is a predicate $univ(e_1, e_2, a)$ where $(e_1, e_2) \in S, a \in AP_{Ac}$

Informally, an absence property asserts that a certain event or state never occurs within the cover of a scope. A universality property asserts that a certain state always holds within the cover of a scope. Formally, each property is an LTL formula which can be interpreted over the execution traces. See Dwyer et al. (1999) and Puaun and Chechik (2003) for the LTL syntactic formulation of absence and universality properties using scopes, and Leucker and Schallhart (2009) for the formal semantics of LTL formulae. For example, the absence property $abs(A^{\downarrow}, C^{\uparrow}, B^{\uparrow})$ is the LTL formula :

$$\Box(((A \wedge X\neg A) \wedge \Diamond(\neg C \wedge XC)) \Rightarrow \neg(\neg B \wedge XB)\mathcal{U}(\neg C \wedge XC))$$

This property asserts that event $B^{\uparrow}$ never happens between events $A^{\downarrow}$ and $C^{\uparrow}$.

Remark 6. (Restriction on universality properties). Since an event refers to a change in the state of an actuator, it does not make sense to assert that an event occurs at every position within the cover of a scope. Thus, universality properties cannot be defined over actuator events.

3.3 Requirements on the set of scopes

The mining step of our approach consists in finding the properties satisfied by the attack-free execution traces collected from supervisory flows between a supervisor and PLCs. Following Definition 5, properties express constraints on sets of actuator states and events, and are defined using different scopes belonging to a set of scopes S. The set of actuator states and events is fixed and depends on the components in the system. On the other hand, the set of scopes S is not fixed a priori. A standard approach to specification mining would explore all possible scopes. However, as discussed in Section 3.1, this leads to a significant number of redundancies. Our main objective is to avoid such redundancies through a careful pre-selection of the scopes in S such that they do not exhibit overlaps.

We also need to ensure that the scopes in S collectively cover each observed trace and no property is missed. Otherwise, an attack might go unnoticed due to the absence of a constraint on a subsequence of the observed traces. Intuitively, this means that we need to limit redundancies without sacrificing coverage. In the following, we discuss four requirements that match these observations : (i) each scope must cover one pair of time positions, (ii) avoiding unconstrained parts of an observed trace, (iii) avoiding scope redundancies, and (iv) avoiding over-specifications.

Requirement (i) reduces the cover of each scope to only one pair of time positions. The reason behind this requirement lies in the form of the LTL specification patterns (see Definition 5) and the corresponding finite state automaton. When the automaton detects a violation, it enters a deadlock state with no outgoing transitions. This means that all the pairs of time positions occuring after the first violation are ignored. To ensure that all pairs of time positions are considered, we restrict the cover of each scope to only one pair of time positions.

From requirement (ii), the cover of the selected scopes should span all time positions in each observed trace. Otherwise, attacks might be missed if they occur within uncovered time positions. The set of scopes $S = \{(e_1, e_4), (e_4, act^{\downarrow})\}$ does not cover all time positions in the observed trace of Figure 3 since the combined covers of (e_1, e_4) and $(e_4, act^{\downarrow})$ is $\{(1, 2), (2, 4)\}$ which does not include $(0, 1)$.

Redundancies (requirement (iii)) happen when two scopes overlap, i.e when the intersection of their covers is not empty. For instance, in Figure 3, the scope (e_1, e_4) with cover $\{(1, 2)\}$ overlaps with the scope (e_1, e_6) with cover $\{(1, 3)\}$ since their cover intersect on $(1, 2)$. Avoiding redundancies is our main objective since it leads to a reduction in the number of deployed monitors which otherwise overload the operator with redundant information.

Over-specifications (requirement (iv)) occur when the scopes are too broad, i.e their covers unnecessarily span too many time positions within the observed trace. This lead to missing some properties during the mining phase. To illustrate this case, consider the set of scopes $S = \{(act^{\uparrow}, e_1), (e_1, act^{\downarrow})\}$ on Figure 3. Suppose that a property

p holds on the pair of time positions $(1,3)$ but not on $(3,4)$. Then p does not hold on $(1,4)$. Since the cover of $(e_1, act^{\downarrow})$ is $\{(1,4)\}$, mining the properties directly on S would miss the fact that p holds on $(1,3)$. To avoid this issue, the scopes need to be of maximal precision, i.e their cover must span the closest time positions possible.

We now give a formal expression of the four requirements.

Proposition 7. A set of scopes S satisfies requirements (i) over a trace t of size n iff $\forall s \in S, |C_t(s)| = 1$.

We assume that the sets of scopes used in the following satisfy Proposition 7.

Proposition 8. Let $I = \{(i,j) \in Dom(t)^2, j - i = 1\}$ be the set of all successive time position pairs in $Dom(t)$. A set of scopes S is non-redundately covering (requirements (ii) and (iii) above) a trace t iff [1] :

$$\forall (i,j) \in I, \exists! s \in S \mid C_t(s) = \{(k,l)\} \wedge k \leq i < j \leq l$$

Proposition 7 ensures that the cover of each scope in S reduces to a single pair of time positions on t. Proposition 8 makes sure that each pair of time positions on t is covered once and only once by a scope in S. For instance, regarding Figure 3, the set of scopes $S = \{(act^{\uparrow}, e_4), (e_1, act^{\downarrow})\}$ satisfies Proposition 7 since $C_t((act^{\uparrow}, e_4)) = \{(0,2)\}$,$C_t((e_1, e_6)) = \{(1,4)\}$ and thus $|C_t((e_1, e_4))| = |C_t((e_1, e_6))| = 1$. However, S does not satisfy Proposition 8 since $(1,2)$ is covered by two scopes : $(act^{\uparrow}, e_4)$ and $(e_1, act^{\downarrow})$.

We now assume that the sets of scopes used in the following satisfy Proposition 7 and 8.

Proposition 9. A set of scopes S is of maximal precision over trace t (requirement (iv)) iff for every other set of scopes S' :

$$\forall (e'_i, e'_j) \in S', \exists (e_i, e_j) \in S \mid$$
$$\begin{cases} C_t((e_i, e_j)) = \{(c_i, c_j)\}, C_t((e'_i, e'_j)) = \{(c'_i, c'_j)\} \\ c'_i \leq c_i < c_j \leq c'_j \end{cases}$$

Proposition 9 says that for any scope (e'_i, e'_j) in a set of scopes S', S contains a scope (e_i, e_j) at least as precise as (e'_i, e'_j). For instance, consider the two sets of scopes $S = \{(act^{\uparrow}, e_4),\ (e_4, act^{\downarrow})\}$ and $S' = \{(act^{\uparrow}, act^{\downarrow})\}$ on Figure 3. Both sets satisfy Propositions 7 and 8. We have $C_t((act^{\uparrow}, act^{\downarrow})) = \{(0,4)\}$, $C_t((act^{\uparrow}, e_4)) = \{(0,2)\}$, and $C_t((e_4, act^{\downarrow})) = \{(2,4)\}$. Since the time position pairs $(0,2)$ and $(2,4)$ are within $(0,4)$, the scopes in S are more precise than S' on $(0,2)$.

3.4 Generation of the set of scopes

In Section 3.3, we discussed the requirements which a set of scopes needs to meet. In this section, we show how such a set of scopes can be generated first for a single observed trace, then for a set of observed traces. To do so, we introduce the notion of a *path*. A path is an application which uniquely associates each time position in the observed trace with a single event. This event must not appear anywhere else in the observed trace, i.e it must characterize the position. The goal of a path is to uniquely characterize every time position in the observed with an event. Using these characterizing events, we will show that a set of scopes satisfying the requirements in Section 3.3 exists. In Figure 3, a possible path derived from the observed trace associates $act^{\uparrow}$ to position 0, e_1 to position 1, e_4 to position 2, e_6 to position 3, and $act^{\downarrow}$ to position 4. Given this path, e_1 uniquely identifies position 1, i.e e_1 is never observed at any other position in the observed trace. Events e_4 and e_6 perform a similar task for positions 1 and 2. On the other hand, position 2 cannot be associated with event e_5 since e_5 occurs also in position 3.

Definition 10. (Path). A *path* p_t over t is an application : $p_t : Dom(t) \rightarrow E$ such that :

$$\begin{cases} \forall i \in Dom(t), p_t(i) \in t(i) \\ \forall (i,j) \in Dom(t)^2, i \neq j \implies p_t(i) \notin t(j) \end{cases}$$

Given an observed trace t, we denote by Π_t the set of all possible paths over t.

Remark 11. (Existence of a path over an observed trace). Generally, one cannot guarantee the existence of a path over an observed trace. A path cannot be generated if one of the time positions can only be assigned events appearing elsewhere on the observed trace. In this case, the position cannot be uniquely characterized. Later, we discuss a solution to this issue which isolates and treats separately the events appearing in the conflicting positions. In the worst case, when no position can be distinguished within the observed trace, our solution reduces to a standard specification mining approach which explores all possible scopes and results in redundancies.

Remark 12. (Number of paths over an observed trace). Given an observed trace t, the number of paths $|\Pi_t|$ can be greater than 1. In the previous example, position 1 could have been associated either with e_1 or e_2. Thus, different paths are compatible with the observed trace in Figure 3.

In what follows, we refer to the image of a path p_t over t as $Img(p_t)$.

Remark 13. (Path bijection). From Definition 10, p_t defines a bijective application from $Dom(t)$ to $Img(p_t)$ since each time position in $Dom(t)$ is associated with a unique event in $Img(p_t)$.

Proposition 14. Given an observed trace t a set of scopes S satisfying Propositions 7, 8, and 9 exists if a derived path exists.

Proof. [**Sketch**] Let t be an observed trace and $p_t \in \mathbb{P}_t$ a derived path. We set $S = \{(p_t(0), p_t(1)), (p_t(1), p_t(2)), \cdots, (p_t(n-2), p_t(n-1))\}$. We show that S satisfies Prop. 7, 8 and 9.

Prop. 7 : Let $(p_t(i), p_t(j)) \in S$. From Remark 13, we know that each p_t defines a bijective mapping from $Dom(t)$ to $Img(p_t)$. Thus, $(p_t(i), p_t(j))$ is uniquely associated to the pair $(i,j) \in Dom(t)^2$. Then, using Definition 3, $C((p_t(i), p_t(j))) = \{(i,j)\}$ and thus $\forall s \in S, |C(s)| = 1$.

Prop. 8 : Let $(i,j) \in I$ be a pair of successive time positions in $Dom(t)$ as defined in Prop 8. By construction, the scopes in S cover all time positions in $Dom(t)$. Thus, there exists a scope in S, namely $(p_t(i), p_t(j))$, covering (i,j). This scope is unique due to the bijectivity of p_t.

Prop. 9 : Let S' be another set of scopes satisfying Prop. 7 and 8. Let $(e'_1, e'_2) \in S'$ and (c'_1, c'_2) the unique pair of time

[1] $\exists!$ means *there is one and only one*

positions (by Prop 7) to which $C_t((e'_1, e'_2))$ reduces. Since, by construction, S covers all successive time positions in $Dom(t)$, we can always map (e'_1, e'_2) to a scope $(e_1, e_2) \in S$ with $C_t((e_1, e_2)) = \{(c_1, c_2)\}$ such that $c'_1 \leq c_1 < c_2 \leq c'_2$.

To find $\mathbb{P}_t$, *i.e.* the set of all possible paths of the observed trace t, we identify, for each time position j in t, the set of events $E_t(j)$ which do not happen elsewhere on t (see Definition 10). Then, to construct $\mathbb{P}_t$, we generate all applications p_t such that $\forall j \in Dom(t), p_t(j) \in E_t(j)$.

Up to this point, we have only considered how to generate a set of scopes S satisfying requirements i)-iv) from a single observed trace (Proposition 14). However, we generally have several observed traces each corresponding to a run of an activity. To generate S for a set of observed traces $T = \{t_1, t_2, \cdots, t_n\}$ using Proposition 14, we rely on the corresponding set of possible derived path sets $\Pi_T = \{\mathbb{P}_{t_1}, \mathbb{P}_{t_2}, \cdots, \mathbb{P}_{t_n}\}$ as shown in Figure 4. Then, the scope identification step determines whether a unique derived path can be obtained from the set Π_T, and if so derives the mining scopes using Proposition 14.

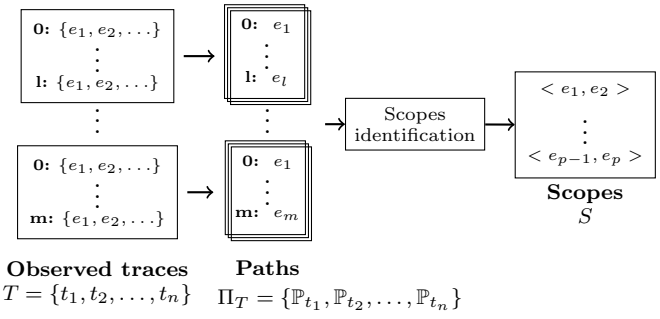

Fig. 4. Scope selection approach based on paths derived from the observed traces

Remark 15. Given a set of observed traces $T = \{t_1, t_2, \cdots, t_n\}$ and its set of possible derived paths sets $\Pi_T = \{\mathbb{P}_{t_1}, \mathbb{P}_{t_2}, \cdots, \mathbb{P}_{t_n}\}$, a non-redundately covering set of scopes of maximal precision exists if :

$$\bigcap_{1 \leq i \leq n} \mathbb{P}_{t_i} \neq \emptyset$$

Remark 15 covers the case when a common path exists among the observed traces. Next, we consider the generation of a set of scopes for the situation where a common path cannot be found. Our solution consists in identifying a minimal set of events, called *conflicting* events, whose removal allows for the generation of a common path. Then, the conflicting events are used with a standard mining approach with no guarantee of non-redundancy while the common path, free of conflicting events, can generate a non-redundant set of scopes.

The main drawback of this solution is the increase in the number of redundant alerts. In the worst case, if all events are conflicting, our solution reduces to a standard mining approach with redundancies. However, due to the regularity of the observed traces generated from activities, we expect the set of conflicting events to be minimal. As we discuss in our evaluation analysis, substantial improvements can still be gained in terms of redundancy compared to a standard approach. In the following, we explore two exhaustive cases where the determination of a common path is impossible. For each case, we discuss how to : (i) identify the set of conflicting events and, (ii) decide on a common path to generate a non-redundant set of scopes.

Case 1. $(\exists \mathbb{P}_t \in \Pi_T \mid \mathbb{P}_t = \emptyset)$. This case corresponds to the situation where a path cannot be generated for an observed trace t, i.e one or more positions can only be assigned events appearing elsewhere in the trace. Let CP_t be the set of such positions, then $CP_t = \{i \in Dom(t) \mid \forall e \in t(i), \exists j \in Dom(t), j \neq i \wedge e \in t(j)\}$. Note that CP_t can never include the first and last position in t, since these characterize the start and end of an activity and do not appear elsewhere in the observed trace.

We identify the conflicting events as those appearing on the positions in CP_t. The set of conflicting events is given by $CE = \{e \in Img(t) \mid \exists j \in CP_t, e \in t(j)\}$. A path can then be generated from $t_{|Dom(t) \setminus CP_t}$, the restriction of t to $Dom(t) \setminus CP_t$.

Case 2. $((\forall \mathbb{P}_{t_i} \in \Pi_T, \mathbb{P}_{t_i} \neq \emptyset) \wedge \bigcap_{1 \leq i \leq n} \mathbb{P}_{t_i} = \emptyset)$. In contrast to Case 1, we assume that paths can be generated for all the observed traces. However, in this case, due to differences in the sequence of events across several runs of an activity, no common path can be found. To determine a common path, we start from the sets of derived paths : $\mathbb{P}_{t_1}, \mathbb{P}_{t_2}, \cdots, \mathbb{P}_{t_n}$. We then identify, for each observed trace t_i, the minimal set of positions CP_i whose events can be removed from a path in $\mathbb{P}_{t_i}$ to generate a common path. Thus, the sets CP_i are determined such that :

$$\exists p_1 \in \mathbb{P}_{t_1}, p_2 \in \mathbb{P}_{t_2}, \cdots, p_n \in \mathbb{P}_{t_n} \mid$$

$$p_{1|Dom(t_1) \setminus CP_1} = p_{2|Dom(t_2) \setminus CP_2} = \cdots = p_{n|Dom(t_n) \setminus CP_n}$$

Where $p_{i|Dom(t_i) \setminus CP_i}$ is the restriction of path p_i to $Dom(t_i) \setminus CP_i$. Then, the set of conflicting events is given by $CE = \bigcup_{p_i} \{e \in Img(p_i) \mid \exists j \in CP_i, e = p_i(j)\}$. A common path is given by $p_{1|Dom(t_1) \setminus CP_1}$.

To generate CP_i for each trace t_i, we compute the longest common subsequence (see Irving and Fraser (1992)) that can be found between any paths $(p_1, p_2, \cdots, p_n)$, $p_i \in \mathbb{P}_{t_i}$. The longest common subsequence finds a common subsequence across the paths by discarding a minimal set of positions CP_i in each path p_i.

After determining the set of scopes S, we generate all the valid *absence* and *universality* properties over all the scopes in S, as indicated in Definition 5. Then, we keep all the properties that are valid on the attack-free training dataset. If conflicting events exist, we use a standard mining approach and only retain valid properties which involve a conflicting event in their scope.

4. EVALUATION

We evaluate our approach, on a hardware-in-the-loop testbed with a simulated physical process controlled by real PLCs. The process is a chemical plant producing benzene by hydrodealkylation of toluene (see Turton (2012)).

The physical process is simulated using OpenModelica [2]. Its parameters (tank dimensions, heating temperatures,

[2] `https://openmodelica.org`

mixing time, etc.) are set so that the physical process undergoes several cycles during our simulations. Control is distributed using three PLCs (Schneider M340 and M580 and Wago IPC-C6 with additional RTU 750-873). Each PLC sends commands and receives sensor information from the real-time OpenModelica simulation via input/output (I/O) interface cards. Control logics are implemented in SFC. To carry its control logics, a PLC may need to communicate with other PLCs to query sensor states or send commands. HMI associated with each allows the operators to monitor and perform manual interventions.

We now describe the simulated physical process and discuss the implementation.

4.1 Testbed Description

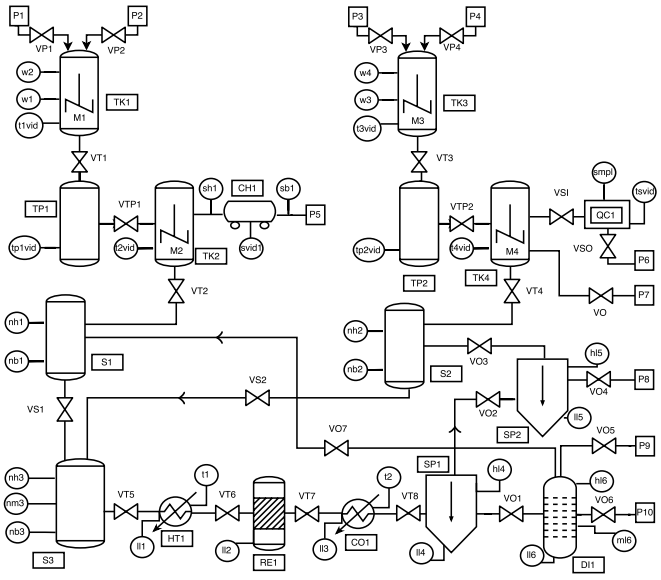

Fig. 5. Physical process model used in our evaluation

The simulated physical process is shown in Figure 5. This process takes input products P1 to P5 and yields output products P6 to P10. The main objective of the process consists in carrying a chemical reaction to synthesize product P10 from reactants in the silos S1 and S2. These reactants are manufactured from initial products in several stages involving mixing (motors M1 to M4) and quality testing (QC1). The reaction occurs in the reactor RE1 and residual reactants are recycled by feeding them back to the silos S1 and S2.

The physical process involves 71 sensors and actuators and has two modes : manual and automatic. The manual mode allows the operators to carry interventions on the process. Upon receiving a command for manual mode, the PLCs put the physical process in a stable condition by manipulating actuators. All in all, the physical process is divided into 18 activities : 10 activities for the generation of the reactants in S1 and S2, and 8 activities for the actual chemical reaction and recycling phase.

4.2 Implementation and datasets

Mining, monitoring, and activity recognition are implemented in C++/Phyton. Analysis is performed on an Intel Dual Core i7 2.6 Ghz machine with 16 GB of RAM and Linux kernel 4.4.0. Evaluation uses 4 network captures during which the physical process goes through several cycles for every activity. The fastest activity goes through tens of cycles in the longest network capture (3 hours). A 2-hours long training traffic is used to build the IDS base profiles and generate the scopes used in mining process specifications. Concurrently, we perform a pre-defined set of legitimate operator interventions. The training dataset reflects realistic conditions where certain legitimate behaviors are absent due to the limited training window. For evaluation purposes, we use 3 captures spanning 3 hours each and containing a total 36 sequence attacks. These attacks occur during all the activities and are performed by sending sequences of malicious Modbus commands from supervisor or engineering stations to the PLCs.

5. ANALYSIS

The first step of the mining process is to identify the scopes to be used in the search for valid properties. Table 1 shows the results of this identification step on the 18 activities in the physical process (A1-A18). The table displays the number of SFC steps in each activity, the number of possible scopes given the actuators and sensors involved in the activity, the time elapsed to identify the scopes and the number of identified scopes to be used in the mining phase. The number of possible scopes is the number of possible pairs of events and can be computed using $M = |E \cup Act|^2$ where E is the set of events and Act is the set of the activity's beginning and end markers.

We observe from Table 1 that the number of mining scopes is always greater or equal than the number of SFC steps for each activity. This indicates that, in general, more precise constraints can be mined on the traces at the level of events and scopes compared to the level of SFC steps. Since the scopes are able to represent finer distinctions within the observed traces, they are more suitable to avoid over-specification issues.

Out of the 18 activities, only A1 and A5 exhibited conflicting events. Each time, discarding one event was sufficient to generate the set of scopes. The conflicting events ($tp1vid^{\uparrow}$ and $tp2vid^{\uparrow}$) indicate that buffer tanks *TP1* and *TP2* are empty. These buffer tanks act as intermediate containers between parallel activities for the generation of reactants in, respectively, *TK1*,*TK2*, and *TK3*,*TK4*. Thus, the levels of buffer tanks *TP1* and *TP2* do not depend only on one activity, but is influenced by actions performed at parallel activities. This means that events $tp1vid^{\uparrow}$ and $tp2vid^{\uparrow}$ cannot uniquely characterize a position in the observed traces of each of the parallel activities, which ultimately results in conflicting events.

The second step of the mining process is to find the valid properties on the training dataset using the scopes. Table 1 shows the mining results for all 18 activities. For each activity, the table displays the amount of time needed for the mining process along with the number of mined properties for this paper's approach, the approach in Koucham et al. (2016) and the approach in Lemieux et al. (2015) using Texada [3]

[3] `http://bitbucket.org/bestchai/texada`

Table 1. Mining results

Activity	**A1***	**A2**	**A3**	**A4**	**A5***	**A6**	**A7**	**A8**	**A9**	**A10**	**A11**	**A12**	**A13**	**A14**	**A15**	**A16**	**A17**	**A18**
# SFC Steps	4	1	1	6	4	1	1	3	3	2	11	4	1	1	1	2	1	1
Pos. scopes	324	64	64	484	324	64	64	196	144	36	1444	100	36	100	64	100	64	64
Scopes' id. time (s)	0.044	0.039	0.054	0.052	0.044	0.062	0.039	0.043	0.045	0.046	2.019	0.042	0.041	0.043	0.039	0.037	0.033	0.034
# Scopes	7	2	2	6	7	2	2	3	4	3	19	5	2	2	2	3	2	2
Mining time (s)	0.354	0.029	0.021	0.376	0.387	0.025	0.022	0.489	0.108	0.130	0.045	2.387	0.034	0.012	0.034	0.022	0.062	0.033
# Mined properties																		
This paper	43	4	4	38	35	4	4	10	10	6	237	6	4	3	4	4	4	4
KMH+16	297	8	8	750	196	8	8	38	56	21	5088	17	8	14	8	28	8	8
LPB15	533	8	8	1498	431	8	8	77	155	37	8029	33	8	32	8	41	8	8

Pos. : Possible, id. : identification, KMH+16 : Koucham et al. (2016), LPB15 : Texada, Lemieux et al. (2015)

Compared to other approaches we consistently reduce the number of mined properties. This is especially apparent for long activities such as $A11$ where we only retain 4.67% (KMH+16) and 2.95% (LPB15) of the properties mined by the other approaches. Generally, longer activities involve more actuators and sensors, increasing the number of possible scopes and of possible properties. In our case, for each activity, the number of scopes on which mining is performed is significantly lower than the number of possible scopes. This provides an advantage over other approaches which do not perform any pre-selection of scopes and produce a large amount of overlapping properties.

To evaluate the performance of our intrusion detection approach, we deploy the monitors and run the three evaluation datasets. All attacks were successfully identified across all datasets. We also record false alarms (35% of the reported violations) due to operator interventions not seen in the training dataset. Generally, we observe one violation per operator action. This follows from the precision requirement on the scope set which ensures that violations report the most precise scopes. Maintaining a limited number of alerts is important to avoid overwhelming the security analyst. In contrast, due to the large amount of redundant mined properties, the other approaches produce 10 to 30 times as much number of violated properties for every operator's action during the longer activities. In this case, the analyst needs to sift through a larger number of violations to discrimate attacks from false alarms.

6. CONCLUSION

Detecting process-aware attacks in ICS requires adequate intrusion detection measures which take into account the physical process. In this paper, we present an approach to efficiently mine safety properties on execution traces of the physical process. The mined properties can then be synthesized as monitors that report any violation to the operator. The proposed mining procedure aims at minimizing redundant properties while ensuring maximum coverage and precision over the execution traces. Finally, we analyze the efficiency of our approach through a hardware-in-the-loop evaluation on a complex physical process model subject to attacks and legitimate operator manipulations. The main prospects of this work concern the handling of false positives. In particular, the integration of feedback from the operator on reported violations and of alerts from other IDS might help achieve lower rates of false positives.

REFERENCES

Bauer, A., Leucker, M., and Schallhart, C. (2006). Model-based runtime analysis of distributed reactive systems. In *ASWEC'06*, 10 pp.

Carcano, a., Coletta, a., and al. (2011). A multidimensional critical state analysis for detecting intrusions in SCADA systems. *IEEE Trans. on Industrial Informatics*, 7(2), 179–186.

Caselli, M., Zambon, E., and Kargl, F. (2015). Sequence-aware Intrusion Detection in Industrial Control Systems. In *Proc. 1st ACM Workshop CPSS*, 13–24.

D'Amorim, M. and Roşu, G. (2005). Efficient monitoring of ω-languages. In *Proc. CAV'05*, 364–378.

Dubey, A., Karsai, G., and Mahadevan, N. (2011). Model-based software health management for real-time systems. In *2011 Aerospace Conference*, 1–18.

Dwyer, M.B., Avrunin, G.S., and Corbett, J.C. (1999). Patterns in property specifications for finite-state verification. In *Proc. ICSE'99*.

Irving, R.W. and Fraser, C. (1992). Two algorithms for the longest common subsequence of three (or more) strings. In *Proc. of the 3rd Annual Symposium on Combinatorial Pattern Matching*, 214–229. Springer-Verlag, London.

John, K.H. and Tiegelkamp, M. (2010). *IEC 61131-3: Programming Industrial Automation*. Springer.

Koucham, O., Mocanu, S., Hiet, G., Thiriet, J.M., and Majorczyk, F. (2016). Detecting Process-Aware Attacks in Sequential Control Systems. In *Proc. NordSec'16*.

Larsen, J. (2008). Breakage - Black Hat. online paper. Accessed 2017-10.

Lemieux, C., Park, D., and Beschastnikh, I. (2015). General LTL specification mining. In *Proc. ASE'15*, 81–92.

Leucker, M. and Schallhart, C. (2009). A brief account of runtime verification. *Journal of Logic and Algebraic Programming*, 78(5), 293–303.

Mitchell, R. and Chen, I.R. (2014). Behavior Rule Specification-based Intrusion Detection for Safety Critical Medical Cyber Physical Systems. *IEEE Tran. on Depend. and Sec. Comp.*, 12(1), 16–30.

Pnueli, A. (1977). The temporal logic of programs. In *Proc. SFCS'77*, 46–57. Washington, DC, USA.

Puaun, D.O. and Chechik, M. (2003). On Closure Under Stuttering. *FAC*, 14, 342–368.

Robert T. Marsh, C. (1997). The report to the president's commission on critical infrastructure protection. Technical report, USA.

Turton, R. (2012). *Analysis, Synthesis, and Design of Chemical Processes*. Prentice-Hall international series in engineering. Prentice Hall.